Configuration Manual

MSc Research Project
Artificial Intelligence

Sreelakshmi Sajikumar
Student ID: x23114185

School of Computing
National College of Ireland

Supervisor: Muslim Jameel Syed

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sreelakshmi Sajikumar
Student ID:	x23114185
Programme:	Artificial Intelligence
Year:	2024-2025
Module:	MSc Research Project
Supervisor:	Muslim Jameel Syed
Submission Due Date:	12-12-2024
Project Title:	Emotion Detection in Text: A Comprehensive Analysis Using Classical, Deep Learning, and Transformer-Based Models
Word Count:	866
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sreelakshmi Sajikumar
Date:	10th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Sreelakshmi Sajikumar
x23114185

1 Introduction

This configuration manual describes the hardware and software requirements, implementation procedures and policies of the Emotion Detection project. The project classifies emotions from textual data into six categories: Six emotions to predict: Anger, Fear, Joy, Love, Sadness, and Surprise, with a proposed multi-phase modeling that combines classical machine learning, deep learning, and transformer-based learning. The deployment covers a Streamlit App as well as Gradio Interface.

1.1 Environment Specification and Configuration

Pre-requisites: Python Version: Python 3.11.5 (Installed locally). Installation Link: https://www.python.org/downloads/ IDE: Visual Studio Code (for local development). Installation Link: https://code.visualstudio.com/ Cloud Environment: Google Colab (for running and deploying Gradio applications). Access Link:https://colab.research.google.com/ Streamlit App: Deployed locally using Streamlit for the FLAN-T5 model. Gradio App: Deployed in Google Colab for easy web-based interaction. Python Package Manager: pip (default in Python 3.11.5).

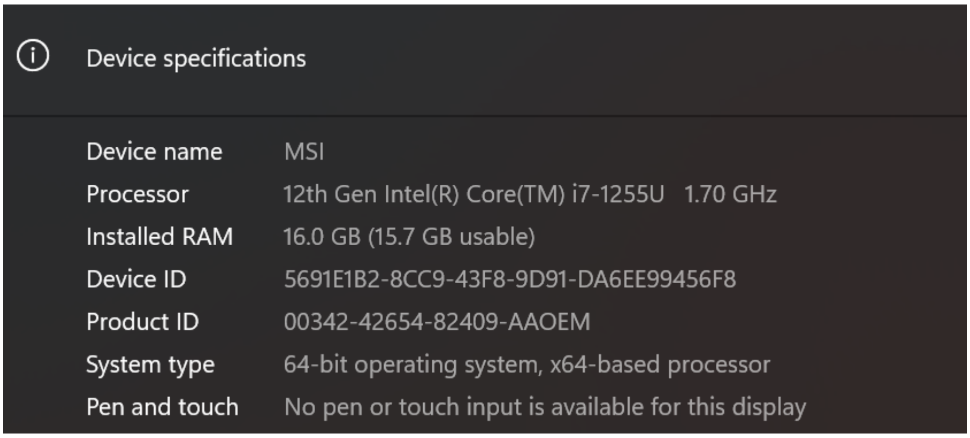
2 System Requirements

2.1 Hardware Requirements

- Device: MSI
- Processor: 12th Gen Intel(R) Core(TM) i7-1255U
- RAM: 16 GB
- Operating System: Windows 11 Home Single Language (64-bit)

2.2 Software Requirements

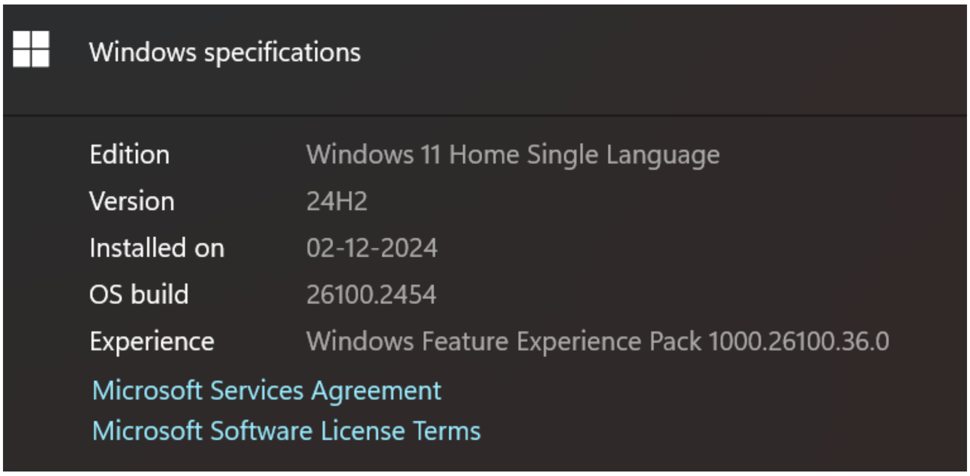
Operating System: Windows 11 Home Single Language Version: 24H2 OS Build: 26100.2454 Experience Pack: 1000.26100.36.0



Device specifications

Device name	MSI
Processor	12th Gen Intel(R) Core(TM) i7-1255U 1.70 GHz
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	5691E1B2-8CC9-43F8-9D91-DA6EE99456F8
Product ID	00342-42654-82409-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Device Specifications



Windows specifications

Edition	Windows 11 Home Single Language
Version	24H2
Installed on	02-12-2024
OS build	26100.2454
Experience	Windows Feature Experience Pack 1000.26100.36.0

Microsoft Services Agreement
Microsoft Software License Terms

Figure 2: Software Requirements

3 Environment Setup

3.1 Virtual Environment Setup

Create a virtual environment for package management python -m venv myenv and Activate the virtual environment

3.2 IDE Configuration

Install Visual Studio Code. And add Python extension for Visual Studio Code from the extensions marketplace. Open project folder in VS Code and set the interpreter the virtual environment.

3.3 Google Colab Configuration

```
from google.colab import files
uploaded = files.upload()
```

Install gradio in colab

```
!pip install gradio
```

3.4 Streamlit Application Deployment

Run the streamlit app locally;

Using the command, streamlit run streamlit_app.py

3.5 Gradio Application Deployment

Use Colab to deploy the Gradio.

```
import gradio as gr

# Define the Gradio interface
def gradio_predict_emotion(text):
    return predict_emotion(text)

# Create the input-output interface
interface = gr.Interface(fn=gradio_predict_emotion,
                     inputs="text",  # Input is a text field
                     outputs="label",  # Output is a label (emotion)
                     title="Emotion Predictor",
                     description="Enter comment to detect the corresponding emotion (Joy, Sadness, Anger, Fear, Surprise, Disgust)")

# Launch the interface
interface.launch()
```

4 Programming Environment Setup

```python
import os
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

Figure 3: Libraries for Data Preprocessing

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from gensim.models import Word2Vec
import nltk
import os
from sklearn.metrics import accuracy_score
```

Figure 4: Libraries for WordtoVect, Glove

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

Figure 5: Libraries for SVM

```
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.layers import LSTM, Embedding, Dense, SpatialDropout1D
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
```

Figure 6: Libraries for LSTM

```
[ ] import pandas as pd
    import os
    import torch
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder
    from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
    from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix
    import matplotlib.pyplot as plt
    import seaborn as sns
```

Figure 7: Libraries for LLM

5 Project Configuration and Execution

5.1 Dataset Preparation

Datasets Used:Twitter Emotion Dataset, Text Emotion Detection Dataset, Emotion Detection Dataset from Kaggle

```
Data :
                                                text    label
0                              i didnt feel humiliated  sadness
1  i can go from feeling so hopeless to so damned...  sadness
2   im grabbing a minute to post i feel greedy wrong    anger
3  i am ever feeling nostalgic about the fireplac...     love
4                                 i am feeling grouchy    anger
```

Figure 8: Data Loading

5.2 Preprocessing Data

Preprocessing: Text cleaning (removal of stopwords, punctuation, and noise). Tokenization and Lemmatization using nltk. Stratified sampling to handle class imbalance.

Execute the dataset preparation notebook (Dataset_Preparation.ipynb) on Google Colab or locally.

Creating a Balanced Dataset

```
['joy' 'love' 'anger' 'surprise' 'fear' 'sadness']
label
joy         14000
love        14000
anger       14000
surprise    14000
fear        14000
sadness     14000
Name: count, dtype: int64
text     84000
label    84000
dtype: int64
```

cleaning and exploring the balanced dataset. It ensures data quality by handling missing and duplicate values. The category distribution analysis and visualization provide valuable insights into the dataset's composition, informing potential further analysis or modeling steps. Saving the cleaned data ensures that the processed dataset is readily available for subsequent tasks.

```
Null value counts:
 text     0
label    0
dtype: int64

Duplicate value counts: 36
dropped the duplicates

Category distribution:
 label
anger       13997
fear        13997
sadness     13997
joy         13993
surprise    13992
love        13988
Name: count, dtype: int64
```

Figure 9: Cleaning and Exploring Dataset

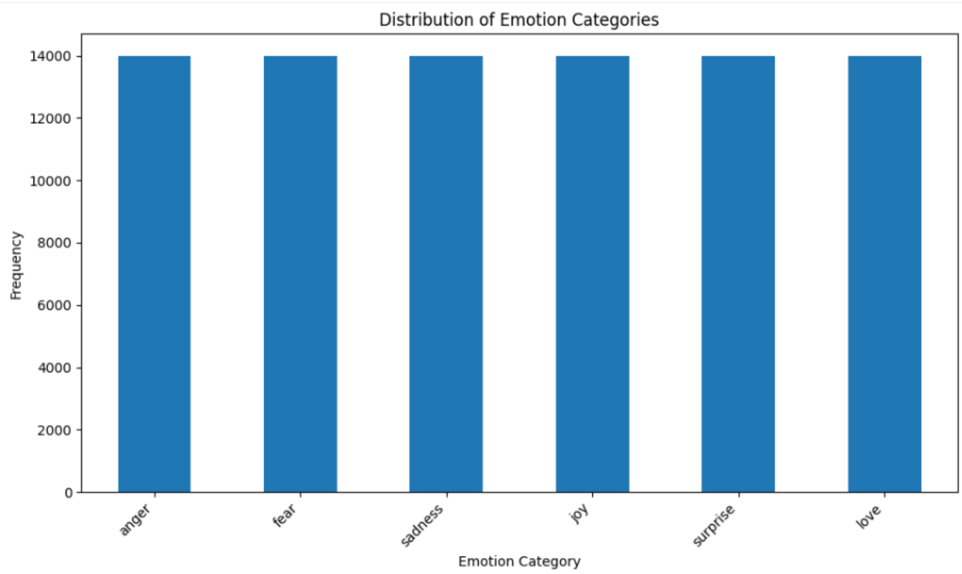


Figure 10: Distribution of Catagories

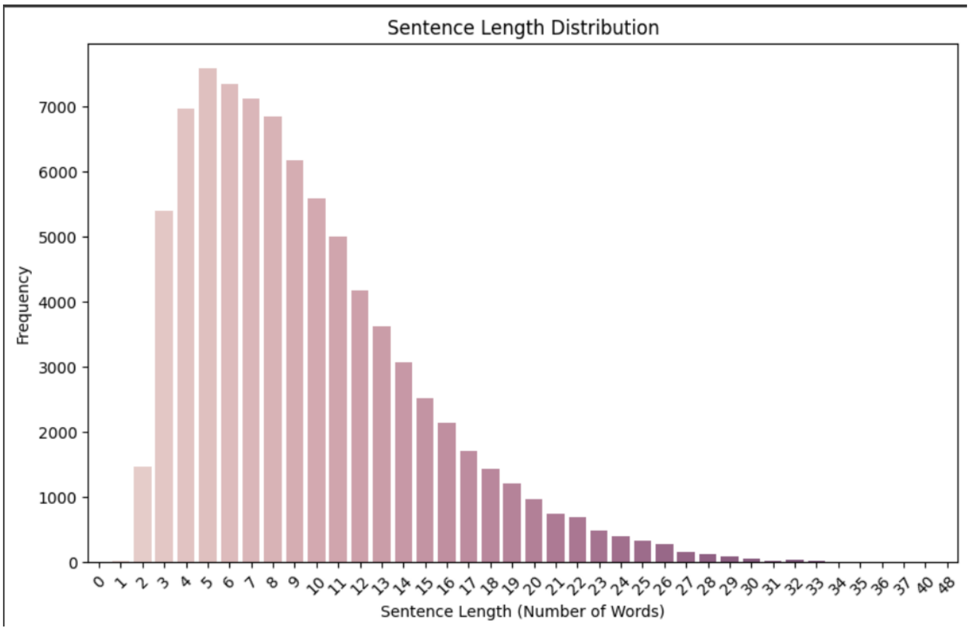


Figure 11: Sentence Length Distribution

5.3 Model Development

Classical Models: Logistic Regression with Bag-of-Words (BoW) features. SVM with Word2Vec and GloVe embeddings. Deep Learning Models: LSTM for sequential data analysis. Bayesian LSTM to handle uncertainty in predictions. Transformer Models: Fine-tuned BERT, RoBERTa, and FLAN-T

Bag of words : The Bag of Words (BoW) model converts text into numerical features by counting the occurrences of words. It disregards grammar and word order, focusing on word frequency.

```
vectorizer = CountVectorizer(max_features=10000)
X_bow = vectorizer.fit_transform(df['preprocessed_text'])
print("BoW Representation Shape:", X_bow.shape)
```

```
BoW Representation Shape: (83960, 10000)
```

```
# Get the feature names (words)
feature_names = vectorizer.get_feature_names_out()

# Convert sparse matrix to dense format for display
X_bow_dense = X_bow.toarray()

# Print feature names and the feature representation for the first document
print("Feature Names:", feature_names)
print("BoW Representation for the first document:", X_bow_dense[0])
```

```
Feature Names: ['aa' 'aaaaall' 'aaron' ... 'zoo' 'zoom' 'zumba']
BoW Representation for the first document: [0 0 0 ... 0 0 0]
```

Figure 12: Bag-of-Words

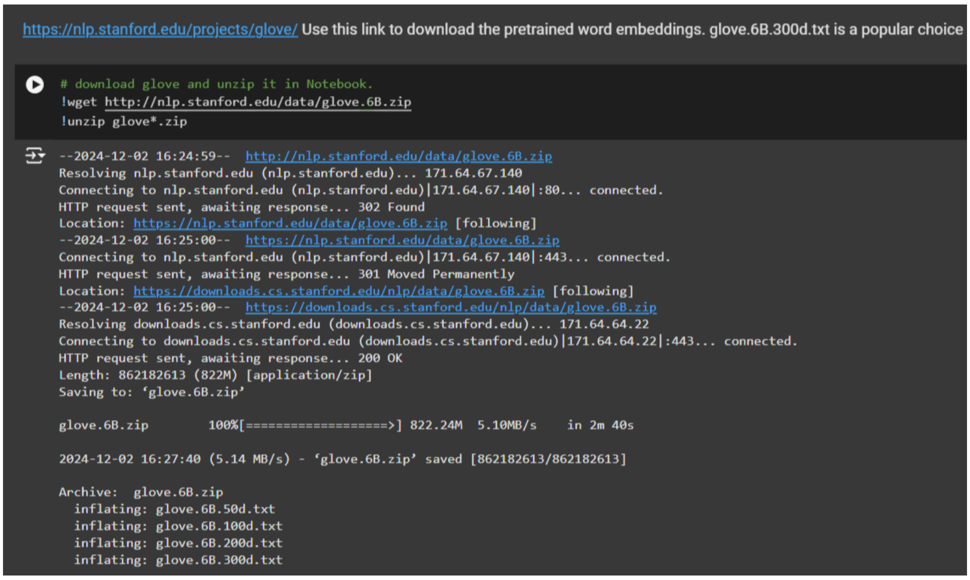
```
# X is the BoW feature matrix
X = X_bow

# Y is the emotion labels
Y = df['label']  # Assuming 'label' column contains the emotion classes

# Split the data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print("Dataset spliting completed")
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

```
Dataset spliting completed
Training data shape: (67168, 10000)
Testing data shape: (16792, 10000)
```

```
[ ] # Train a Word2Vec model
    word2vec_model = Word2Vec(sentences=X_train.tolist(), vector_size=1000, window=5, min_count=1, workers=4, sg=1)

    # Function to convert tokens to document vectors
    def get_document_vector(tokens, model):
        vectors = [model.wv[word] for word in tokens if word in model.wv]
        if len(vectors) > 0:
            return np.mean(vectors, axis=0)  # Average word vectors
        else:
            return np.zeros(model.vector_size)

    # Generate document vectors for training and test data
    X_train_vectors = X_train.apply(lambda x: get_document_vector(x, word2vec_model))
    X_test_vectors = X_test.apply(lambda x: get_document_vector(x, word2vec_model))

    # Convert to numpy arrays
    X_train_vectors = np.stack(X_train_vectors)
    X_test_vectors = np.stack(X_test_vectors)
```

```
WARNING:gensim.models.word2vec:Each 'sentences' item should be a list of words (usually unicode strings). First item here is instead plain <class 'str'>.
```

Figure 13: WordtoVec

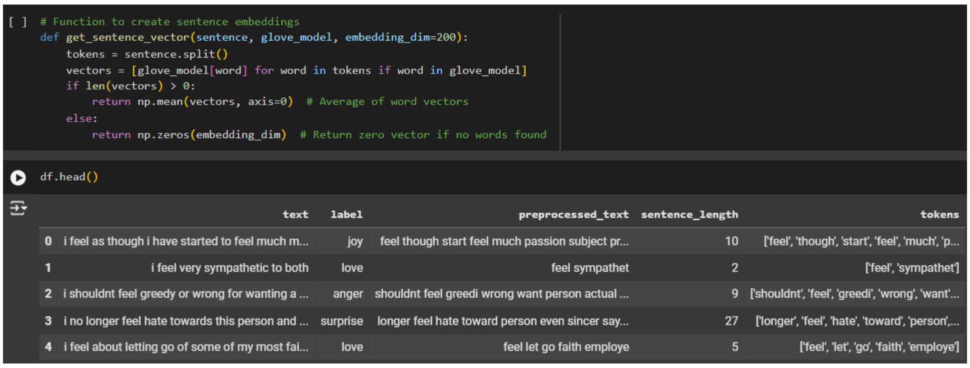


https://nlp.stanford.edu/projects/glove/ Use this link to download the pretrained word embeddings. glove.6B.300d.txt is a popular choice

```
# download glove and unzip it in Notebook.
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove*.zip
```

```
--2024-12-02 16:24:59--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-12-02 16:25:00--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-12-02 16:25:00--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.10MB/s    in 2m 40s

2024-12-02 16:27:40 (5.14 MB/s) - 'glove.6B.zip' saved [862182613/862182613]

Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

Figure 14: Glove



```
# Function to create sentence embeddings
def get_sentence_vector(sentence, glove_model, embedding_dim=200):
    tokens = sentence.split()
    vectors = [glove_model[word] for word in tokens if word in glove_model]
    if len(vectors) > 0:
        return np.mean(vectors, axis=0)  # Average of word vectors
    else:
        return np.zeros(embedding_dim)  # Return zero vector if no words found
```

```
df.head()
```

	text	label	preprocessed_text	sentence_length	tokens
0	i feel as though i have started to feel much m...	joy	feel though start feel much passion subject pr...	10	['feel', 'though', 'start', 'feel', 'much', 'p...
1	i feel very sympathetic to both	love	feel sympathet	2	['feel', 'sympathet']
2	i shouldnt feel greedy or wrong for wanting a ...	anger	shouldnt feel greedi wrong want person actual ...	9	['shouldnt', 'feel', 'greedi', 'wrong', 'want'...
3	i no longer feel hate towards this person and ...	surprise	longer feel hate toward person even sincer say...	27	['longer', 'feel', 'hate', 'toward', 'person',...
4	i feel about letting go of some of my most fai...	love	feel let go faith employe	5	['feel', 'let', 'go', 'faith', 'employe']

Figure 15: Sentence Embedding using Glove

```
# Step 3: Build LSTM Model by training an embedding layer
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))  # Dropout layer to prevent overfitting
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))  # LSTM layer
model.add(Dense(6, activation='softmax'))  # Output layer for 6 classes

# Step 4: Compile the Model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

```
# Step 5: Train the Model
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))

# Step 6: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
Epoch 1/5
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 229s 216ms/step - accuracy: 0.6382 - loss: 0.9747 - val_accuracy: 0.9203 - val_loss: 0.2273
Epoch 2/5
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 211s 201ms/step - accuracy: 0.9299 - loss: 0.1955 - val_accuracy: 0.9276 - val_loss: 0.1938
Epoch 3/5
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 211s 201ms/step - accuracy: 0.9407 - loss: 0.1532 - val_accuracy: 0.9268 - val_loss: 0.2009
Epoch 4/5
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 210s 200ms/step - accuracy: 0.9467 - loss: 0.1351 - val_accuracy: 0.9234 - val_loss: 0.1989
Epoch 5/5
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 213s 203ms/step - accuracy: 0.9512 - loss: 0.1183 - val_accuracy: 0.9213 - val_loss: 0.2109
525/525 ━━━━━━━━━━━━━━━━━━━━ 13s 25ms/step - accuracy: 0.9206 - loss: 0.2111
Test Loss: 0.21089836955070496
Test Accuracy: 0.9212720394134521
```

```
# Predict on the test data
y_pred_probs = model.predict(X_test)  # Get predicted probabilities for each class
y_pred = np.argmax(y_pred_probs, axis=1) # Get the class with the highest probability

# Print the first 10 predictions
print("First 10 Predictions:")
print(y_pred[:10])
```

```
525/525 ━━━━━━━━━━━━━━━━━━━━ 19s 35ms/step
First 10 Predictions:
[4 1 5 0 0 1 2 0 0 2]
```

Figure 16: LSTM for sequential Data Analysis

```
!pip install transformers datasets torch scikit-learn
```

Show hidden output

```
file_path = "data_with_tokens.xlsx"
df = pd.read_excel(file_path)  # Load dataset - Replace with your dataset file
print(df.head())
```

```
                                                text     label  \
0  i feel as though i have started to feel much m...       joy
1                       i feel very sympathetic to both      love
2  i shouldnt feel greedy or wrong for wanting a ...     anger
3  i no longer feel hate towards this person and ...  surprise
4  i feel about letting go of some of my most fai...      love

                                   preprocessed_text  sentence_length  \
0  feel though start feel much passion subject pr...               10
1                                     feel sympathet                2
2  shouldnt feel greedi wrong want person actual ...                9
3  longer feel hate toward person even sincer say...               27
4                         feel let go faith employe                5

                                              tokens
0  ['feel', 'though', 'start', 'feel', 'much', 'p...
1                               ['feel', 'sympathet']
2  ['shouldnt', 'feel', 'greedi', 'wrong', 'want'...
3  ['longer', 'feel', 'hate', 'toward', 'person',...
4          ['feel', 'let', 'go', 'faith', 'employe']
```

Figure 17: Loading Data- Transformer-Based Model

```python
class EmotionDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item["labels"] = torch.tensor(self.labels[idx])
        return item

# Create dataset objects
train_dataset = EmotionDataset(train_encodings, train_labels)
test_dataset = EmotionDataset(test_encodings, test_labels)
```

```python
print("the samples in train dataset : ",train_dataset.__len__())
print("the samples in test dataset : ",test_dataset.__len__())
```

```
the samples in train dataset :  32000
the samples in test dataset :  8000
```

Figure 18: Transform the data into a PyTorch-compatible dataset

```python
# 6. Load Pretrained BERT Model
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=len(label_encoder.classes_)) # len(label_encoder.classes_) is 6 here
# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)  # Move model to the device
```

Figure 19: Build the BERT MODEL

The Google FLAN-T5 model is a fine-tuned version of the T5 (Text-to-Text Transfer Transformer) designed for a variety of NLP tasks, including classification tasks like identifying emotions. Since T5 models treat every NLP problem as a text generation task.

```python
from transformers import pipeline

# Load the zero-shot-classification pipeline with BART Natural Language Inference dataset
zero_shot_classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")

# Sample text (tweet or sentence)
text = "I had such a great day, I feel so happy and energized!"

# Define the candidate labels for emotion detection
labels = ["joy", "anger", "sadness", "fear", "surprise", "disgust"]

# Zero-shot classification
result = zero_shot_classifier(text, candidate_labels=labels)

# Print the result
print("Predicted Emotion:", result['labels'][0], "with score:", result['scores'][0])
```

```
config.json: 100% 1.15k/1.15k [00:00<00:00, 59.5kB/s]
model.safetensors: 100% 1.63G/1.63G [00:20<00:00, 54.2MB/s]
tokenizer_config.json: 100% 26.0/26.0 [00:00<00:00, 512B/s]
vocab.json: 100% 899k/899k [00:00<00:00, 5.63MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 5.66MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 5.05MB/s]
Hardware accelerator e.g. GPU is available in the environment, but no `device` argument is passed to the `Pipeline` object. Model will be on CPU.
Predicted Emotion: joy with score: 0.8782263398170471
```

Figure 20: Zero Shot Prompting using BART

In few-shot prompting, we provide the model with a few examples of how the task should be performed before giving it the new input. This helps the model understand the pattern or task better.

RoBERTa: We'll use RoBERTa, which is a strong model for classification tasks, in a few-shot manner. We'll give a few examples and then ask it to classify the new text into one of the emotion categories.

```python
from transformers import pipeline

# Load RoBERTa model pre-trained for NLI tasks
classifier = pipeline("zero-shot-classification", model="roberta-large-mnli")
tweet_to_classify = "I had such a great day, I feel so happy and energized!"
# Define the prompt template
prompt_template = f"""
You are a helpful assistant that performs sentiment analysis on tweets. Your task is to classify each tweet into one of the following emotions: Joy, Sad, Anger, Fear, Surprise, Disgust.

Here are some examples:

Tweet: "I just got a promotion at work, feeling so accomplished!"
Emotion: Joy

Tweet: "I can't believe my flight got canceled again. This is so frustrating."
Emotion: Anger

Tweet: "The movie I watched last night was incredibly scary."
Emotion: Fear

Tweet: "I feel so down today. Everything seems pointless."
Emotion: Sad

Tweet: "Wow, I wasn't expecting this gift. What a pleasant surprise!"
Emotion: Surprise

Tweet: "The food was so disgusting, I couldn't even finish my meal."
Emotion: Disgust

Now classify the following tweet:

Tweet: "{tweet_to_classify}"
Emotion:
"""

print("prompt template :",prompt_template)


# Candidate labels for emotion detection
labels = ["joy", "anger", "sadness", "fear", "surprise", "disgust"]
```

Figure 21: RoBERTa for Few shot prompting

FLAN-T5: We'll use FLAN-T5, a variant of T5 fine-tuned for instruction-based tasks. It performs well in few-shot settings, where you provide a few examples and a prompt.

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
# Load the FLAN-T5 model and tokenizer
model_name = "google/flan-t5-base"  # You can use other sizes like "flan-t5-large" or "flan-t5-small"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

```python
def create_emotion_prompt(text):
    return f"""
Classify the emotion of the following text into one of these categories: Joy, Sad, Anger, Fear, Surprise, Disgust.

Text: "{text}"

Emotion:
"""
```

Figure 22: Load the Flan-t5 Model

The Google FLAN-T5 model is a fine-tuned version of the T5 (Text-to-Text Transfer Transformer) designed for a variety of NLP tasks, including classification tasks like identifying emotions. Since T5 models treat every NLP problem as a text generation task, you can adapt them to classify emotions by providing an appropriate prompt.

```
[ ]  # Load the DistilBERT tokenizer
     model_name = "distilbert-base-uncased"
     tokenizer = AutoTokenizer.from_pretrained(model_name)

     # Tokenize the dataset
     def tokenize_function(texts):
         return tokenizer(texts, padding=True, truncation=True, max_length=128)

     train_encodings = tokenize_function(train_texts)
     test_encodings = tokenize_function(test_texts)
```

Figure 23: Load the DistilledBERT

6 Deployment

6.1 Streamlit Deployment

Save the streamlit_app.py file in the local project folder. Run the app using the command streamlit run streamlit_app.py Input sample text to get predicted emotions.

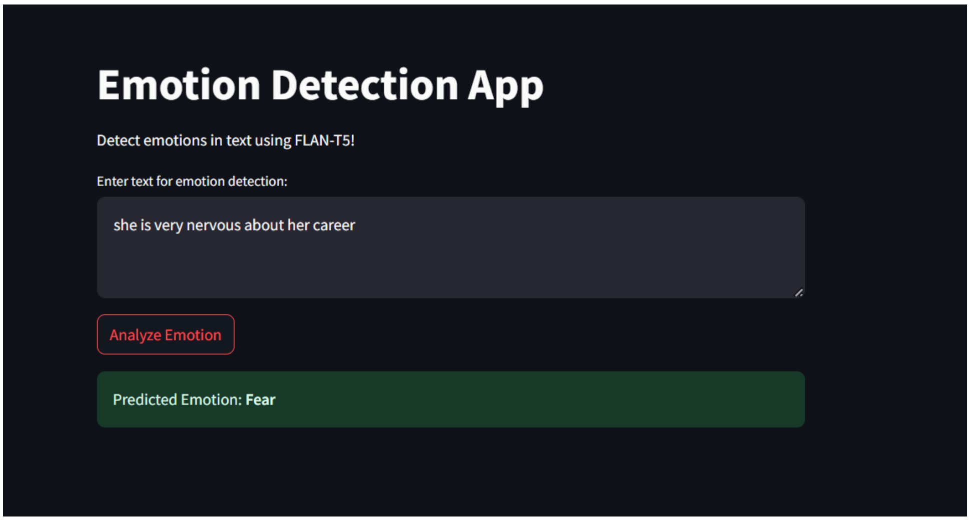


Figure 24: Emotion Detection App

6.2 Gradio Deployment

Purpose: To provide an accessible web-based interface for emotion detection using Gradio. Steps: Upload the Gradio deployment notebook (Gradio_Deployment.ipynb) to Google Colab. Install Gradio using pip install gradio. Run the notebook to launch a Gradio interface with a public URL for real-time interaction.

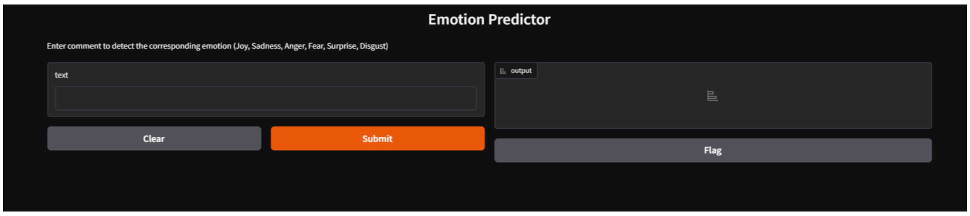


Figure 25: Emotion Predictor using Gradio

7 Evaluation

Metrics Used: Accuracy, Precision, Recall, F1-Score, Confusion Matrix.

7.1 Logistic Regression

```
[16] # Make predictions on the test set
     y_pred = lr_clf.predict(X_test)
     # Evaluate the model
     print(classification_report(y_test, y_pred))

                  precision    recall  f1-score   support

           anger       0.89      0.90      0.89      2819
            fear       0.86      0.85      0.85      2738
             joy       0.85      0.83      0.84      2809
            love       0.87      0.87      0.87      2839
         sadness       0.90      0.87      0.89      2792
        surprise       0.88      0.93      0.90      2795

        accuracy                           0.88     16792
       macro avg       0.88      0.88      0.88     16792
    weighted avg       0.88      0.88      0.88     16792
```

Figure 26: Logistics Regression Evaluation Metrics

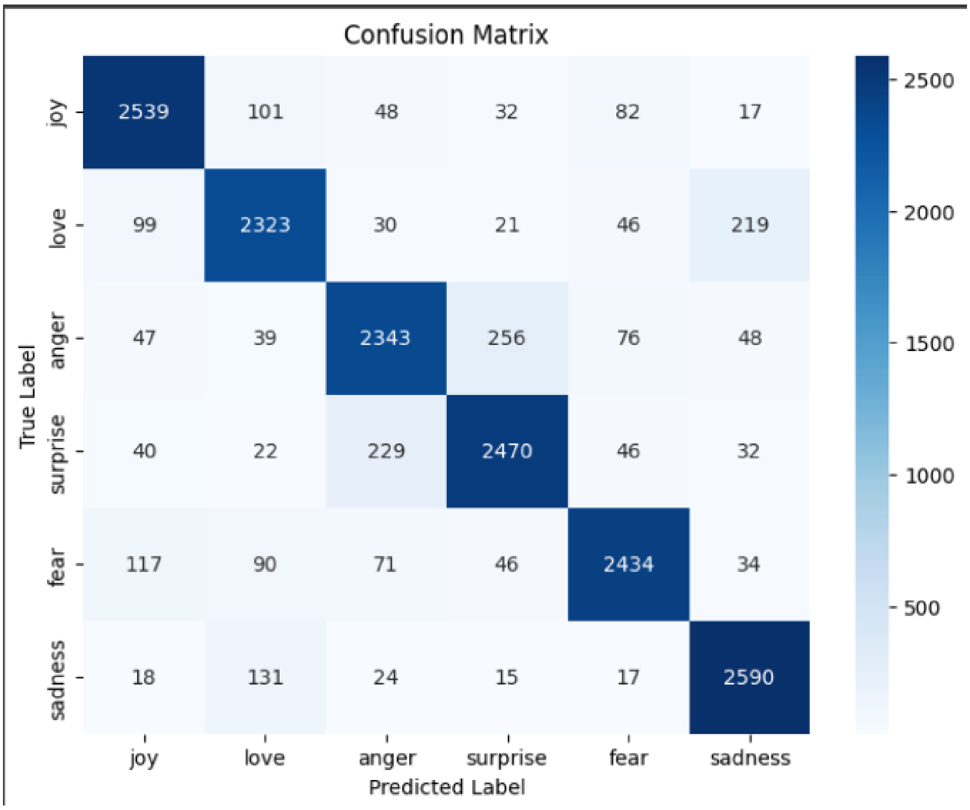


Figure 27: Logistic Regression Confusion Metrics

7.2 Support Vector Machine

```
# Train an SVM classifier
svm_model = SVC(kernel='rbf', random_state=42, gamma=0.1)
svm_model.fit(X_train_vectors, y_train)

# Make predictions

y_pred = svm_model.predict(X_test_vectors)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

```
# Assuming y_test is the true labels and y_pred are the predicted labels
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy score
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.23
```

Figure 28: SVM Evaluation Metrics

7.3 LSTM

	precision	recall	f1-score	support
anger	0.92	0.94	0.93	2819
fear	0.92	0.88	0.90	2738
joy	0.90	0.89	0.90	2809
love	0.91	0.92	0.91	2839
sadness	0.97	0.92	0.94	2792
surprise	0.91	0.97	0.94	2795
accuracy			0.92	16792
macro avg	0.92	0.92	0.92	16792
weighted avg	0.92	0.92	0.92	16792

Figure 29: LSTM Evaluation Metrics

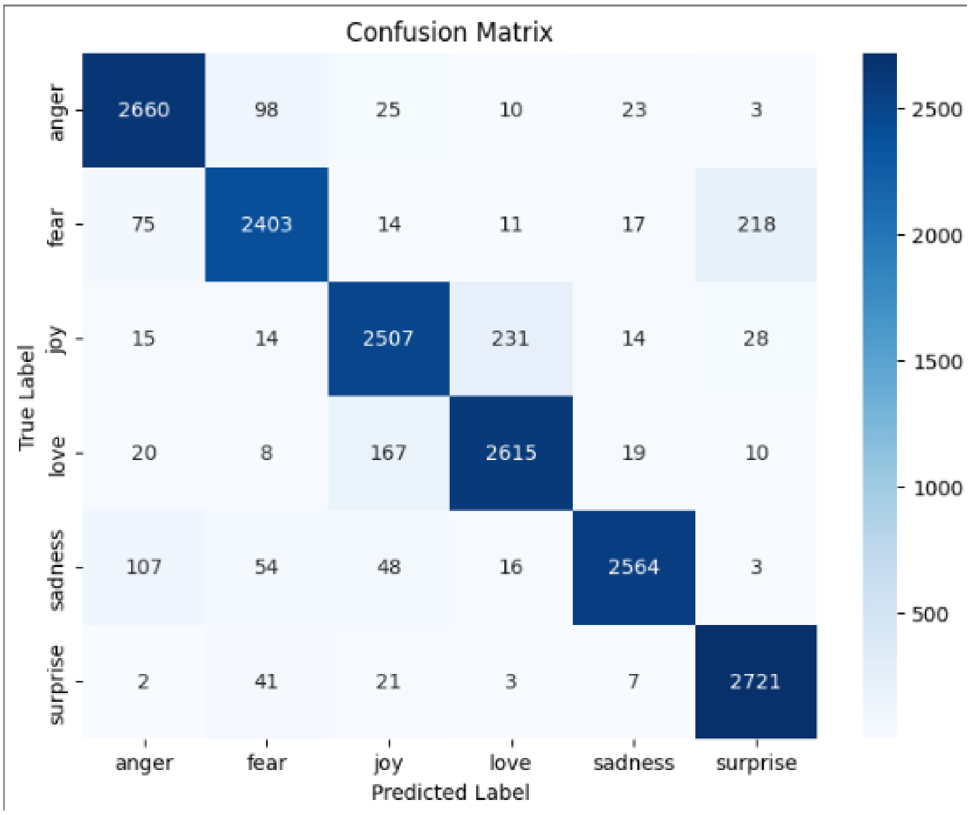


Figure 30: LSTM Confusion Metrics

7.4 LLM

```
Classification Report:
              precision    recall  f1-score   support

       anger       0.91      0.94      0.93      1346
        fear       0.94      0.86      0.90      1315
         joy       0.93      0.83      0.88      1324
        love       0.88      0.93      0.91      1320
     sadness       0.93      0.92      0.92      1338
    surprise       0.90      1.00      0.94      1357

    accuracy                           0.91      8000
   macro avg       0.91      0.91      0.91      8000
weighted avg       0.91      0.91      0.91      8000

('./bert-emotion/tokenizer_config.json',
 './bert-emotion/special_tokens_map.json',
 './bert-emotion/vocab.txt',
 './bert-emotion/added_tokens.json',
 './bert-emotion/tokenizer.json')
```

Figure 31: LLM Evaluation Metrics

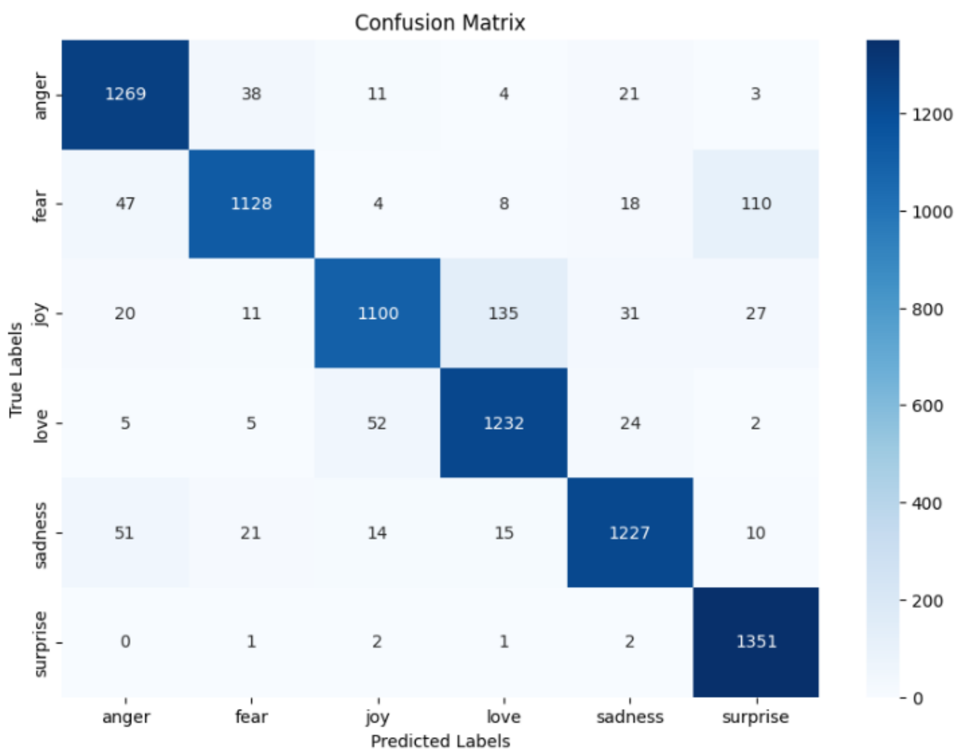


Figure 32: LLM Confusion Metrics

8 Conclusion

The project was accurately able to show how emotion detection can be done through classical, deep learning as well as transformer models. There were real-life use cases demonstrated through Streamlit and Gradio apps with regards to customer feedback analysis, sentiment management, and mental health. The future work includes the collection of more diverse datasets, the scaling up of the transformer model, and the addition of methods for explaining the model to the user.